Table of contents

Adds support for migrations using JAXB classes

Grails JAXB Database Migration Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 0.1

Table of Contents

1 Introduction to the Database Migration JAXB Plugin

The Database Migration JAXB plugin extends the Database Migration plugin to add the ability to write changelogs in Java or Groovy code using JAXB classes.

The plugin uses JAXB 2.2 classes, so you must either use Java 7 or higher, or upgrade to the newer version of JAXB. I don't know if this is possible since the classes are part of the JDK, so your best bet is to use Java 7+.

1.1 History

History

- January 5, 2013
    - initial 0.1 release

2 Getting Started

The first step is to add a dependency for the plugin in `BuildConfig.groovy`:

```
plugins {
   …
   runtime ':database-migration-jaxb:0.1'
}
```

This should pull in the `database-migration` plugin transitively but you can also explicitly add in a dependency for that, in particular if you want to use a newer version than the plugin requires:

```
plugins {
   …
   runtime ':database-migration:1.3.2'
}
```

3 Usage

You can still create changelogs in traditional XML Liquibase format, or use Groovy-based scripts, but using this plugin adds an additional option of using JAXB-based scripts. The top-level file must be a Groovy file, e.g. `grails-app/migrations/changelog.groovy`, but you can include JAXB-based classes with syntax like this:

```
include file: 'com.yourcompany.yourapp.SomeClassName.class'
```

Include the full class name including package. The "file" name must end in ".class" so the correct parser is used.

Class structure

Classes can be written in Java or Groovy. They should implement the `grails.plugin.databasemigration.jaxb.JaxbChangelogGenerator` interface:

```
public interface JaxbChangelogGenerator {
   List<DatabaseChangeLog> generate();
}
```

This isn't required however; your class just needs a no-arg method named `generate` that returns a List of one or more `org.liquibase.xml.ns.dbchangelog.DatabaseChangeLog` instances.

For the most part the API of the JAXB classes should be clear; read the source if not. They were generated from the schema that is used for Liquibase XML files (and therefore the Groovy DSL since it's based on the XML syntax) so the same attributes are supported.

Here's an example of a class written in Java; this is from the plugin's integration tests:

```
package grails.plugin.databasemigration.jaxb;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.liquibase.xml.ns.dbchangelog.Column;
import org.liquibase.xml.ns.dbchangelog.Constraints;
import org.liquibase.xml.ns.dbchangelog.CreateTable;
import org.liquibase.xml.ns.dbchangelog.DatabaseChangeLog;
import org.liquibase.xml.ns.dbchangelog.DatabaseChangeLog.ChangeSet;

public class CreateDepartmentTable implements JaxbChangelogGenerator {

public List<DatabaseChangeLog> generate() {

ChangeSet changeSet = new DatabaseChangeLog.ChangeSet();
      changeSet.setId("department");
      changeSet.setAuthor("Bob");

CreateTable createTable = new CreateTable();
      createTable.setTableName("department");

Column column = new Column();
      column.setName("id");
      column.setType("int");
      List<Object> columnContent = new ArrayList<Object>();
      Constraints constraints = new Constraints();
      constraints.setPrimaryKey("true");
      constraints.setNullable("false");
      columnContent.add(constraints);
      column.setContent(columnContent);
      createTable.getColumn().add(column);

column = new Column();
      column.setName("name");
      column.setType("varchar(50)");
      createTable.getColumn().add(column);

column = new Column();
      column.setName("active");
      column.setType("boolean");
      column.setDefaultValueBoolean("true");
      columnContent = new ArrayList<Object>();
      constraints = new Constraints();
      constraints.setNullable("false");
      columnContent.add(constraints);
      column.setContent(columnContent);
      createTable.getColumn().add(column);

changeSet.getChangeSetChildren().add(createTable);

DatabaseChangeLog databaseChangeLog = new DatabaseChangeLog();
      databaseChangeLog.getChangeSetOrIncludeOrIncludeAll().add(changeSet);

return Collections.singletonList(databaseChangeLog);
   }
}
```

This class is nearly identical except that it's written in Groovy:

```
package grails.plugin.databasemigration.jaxb

import org.liquibase.xml.ns.dbchangelog.Column
import org.liquibase.xml.ns.dbchangelog.Constraints
import org.liquibase.xml.ns.dbchangelog.CreateTable
import org.liquibase.xml.ns.dbchangelog.DatabaseChangeLog
import org.liquibase.xml.ns.dbchangelog.DatabaseChangeLog.ChangeSet

class CreateDepartment2Table {

def generate() {

def changeSet = new DatabaseChangeLog.ChangeSet(id: 'department2', author:
'Bob')

def createTable = new CreateTable(tableName: 'department2')
      createTable.column << new Column(name: 'id', type: 'int', content: [
new Constraints(primaryKey: true, nullable: false)])
      createTable.column << new Column(name: 'name', type: 'varchar(50)')
      createTable.column << new Column(name: 'active', type: 'boolean',
defaultValueBoolean: true, content: [new Constraints(nullable: false)])
      changeSet.changeSetChildren << createTable

[new DatabaseChangeLog(changeSetOrIncludeOrIncludeAll: [changeSet])]
   }
}
```

It uses a different change set id since they must be unique, and a slightly different table name so they can be run in the same test. It also doesn't implement the `JaxbChangelogGenerator` interface, although it's a good idea to do so to make sure the signature is correct. Obviously the code is dramatically more compact and in general classes should probably be written in Groovy to not be overly verbose.